Administración de Servidores

Memoria Práctica 1: Linux Básico

Autores:
Alberto Aragón Torralba
Alejandro Guerrero Medina
Gonzalo Sánchez Bazán

Grado en Ingeniería Informática
Curso 2022 - 2023

Índice

1. Preparación del entorno **2**
1.1. Ejercicio 1 . . . 2

2. Órdenes básicas **3**
2.1. Ejercicio 2 . . . 3
2.2. Ejercicio 3 . . . 3
2.3. Ejercicio 4 . . . 3
2.4. Ejercicio 5 . . . 4

3. Procesos **6**
3.1. Ejercicio 6 . . . 6
3.2. Ejercicio 7 . . . 6
3.3. Ejercicio 8 . . . 7

4. Gestión de usuarios **8**
4.1. Ejercicio 9 . . . 8
4.2. Ejercicio 10 . . . 8
4.3. Ejercicio 11 . . . 9

5. Sistema de ficheros **10**
5.1. Ejercicio 12 . . . 10
5.2. Ejercicio 13 . . . 10
5.3. Ejercicio 14 . . . 11

6. Bash scripting **12**
6.1. Ejercicio 15 . . . 12
6.2. Ejercicio 16 . . . 12
6.3. Ejercicio 17 . . . 13
6.4. Ejercicio 18 . . . 15

1. Preparación del entorno

1.1. Ejercicio 1

Primero debemos crear una máquina Ubuntu con **Vagrant**. En este documento usaremos como referencia la *box*: **ubuntu/bionic64**, pero se puede usar cualquier otra que sea, al menos, una máquina Ubuntu.

En una terminal de comandos, deberemos ejecutar el siguiente comando:

```
vagrant init ubuntu/bionic64
```

Con lo que **Vagrant** iniciará una máquina con la *box* indicada, junto con su respectivo *Vagrantfile*, donde configuraremos los siguientes aspectos:

- La configuración red indicada en el enunciado, donde el puerto **80** de la máquina invitada es redireccionado al puerto **8080** del anfitrión.
- *Apache* instalado en la máquina, donde el provisioner *shell* ejecuta la configuración inicial de la máquina.

En esencia, el fichero *Vagrantfile*, obviando la configuración por defecto, debería de contener las siguientes instrucciones añadidas o modificadas:

```
Vagrant.configure("2") do |config|
    config.vm.box = "ubuntu/bionic64"
    config.vm.network "forwarded port", guest: 80, host: 8080
    config.vm.provision "shell", inline: <<-SHELL
        apt-get update
        apt-get install -y apache2
    SHELL
end
```

Una vez preparada la máquina con su respectiva configuración, ejecutando las lineas de comando:

```
vagrant up
vagrant ssh
```

Creamos la máquina acorde a la configuración del *Vagrantfile* asociado y nos conectaremos por *ssh* a la máquina invitada, ya preparada para poder trabajar en ella.

VALIDACIÓN: Realizamos tres comprobaciones esenciales en el *script*:

- Que la máquina Ubuntu se encuentre instalada –comprobado mediante el comando grep buscando '`ubuntu`' en el fichero *Vagrantfile*–
- Que se encuentre *Apache* instalado –ídem a la comprobación previa, buscando '`apache2`'–
- Que esté el puerto **80** redirigido –ídem a la comprobación previa, buscando '`config.vm.network "forwarded_port", guest: 80`'–

2. Órdenes básicas

2.1. Ejercicio 2

ENUNCIADO: **Muestre todos los ficheros de C (.c, .h) del sistema.**
SOLUCIÓN:

```
sudo find / -type f \( -name "*.c" -o -name "*.h" \)
```

EXPLICACIÓN: El comando `find` resulta ser ideal en casos donde se requiera hacer una búsqueda dentro del sistema para localizar archivos de un tipo en directorios asociados, tal y como ocurre en el enunciado especificado. En su semántica, encontramos:

- `sudo`: Para evitar mostrar avisos referidos a *permiso denegado a fichero*.
- `type -f`: Indicamos que el tipo de archivo que estamos buscando es de tipo **común**.
- `\( -name` "`*.c`" `-o -name` "`*.h`" `\)`: Expresión regular en la que indicamos mediante las *flags* `-name` todo archivo que sea `.c` o `.h`.

VALIDACIÓN: Hacemos un pequeño campo de prueba con un directorio en el que crearemos una serie de archivos de los que se encuentran `.c` y `.h`, entre otros. Luego, realizamos el mismo comando en el *script* con una funcionalidad añadida para contar líneas y en caso de que coincidan con el número de ficheros creados –en este caso, 6, pero se deberán contar solamente 4 acorde al criterio–, mostrará el mensaje pertinente, ya sea de éxito o error.

2.2. Ejercicio 3

ENUNCIADO: **Modifique la solución del ejercicio anterior para que no se muestren los errores.**
SOLUCIÓN:

```
sudo find / -type f \( -name "*.c" -o -name "*.h" \) 2> /dev/null
```

EXPLICACIÓN: Para este ejercicio, lo único que tendremos que añadir a la sentencia anterior será la expresión `2>`, lo que hace redireccionar el flujo estándar de errores (`stderr`) al fichero indicado; `/dev/null`, un fichero especial cuyo propósito es descartar toda información contenida.
VALIDACIÓN: En pocas palabras, redireccionamos la salida del comando `ls` al fichero `\dev\null` y mostramos el contenido. Eventualmente, no habrá nada.

2.3. Ejercicio 4

ENUNCIADO: **Muestre por pantalla cuántos usuarios no pueden iniciar sesión.**
SOLUCIÓN:

```
grep -F 'nologin' /etc/passwd | cut -d: -f1
```

EXPLICACIÓN: El comando `grep` busca un patrón definido en un archivo de texto. Es una de las herramientas más versátiles y útiles que ofrece *Unix* junto con el comando `find`. Veamos con más detalle el modo de empleo en este enunciado:

- `grep -F 'nologin' /etc/passwd:` Especificamos que se interpreten los patrones como cadenas literales, en este caso `nologin`, en el fichero `/etc/passwd`.
- `cut -d: -f1:` El comando `cut` elimina secciones del fichero especificado que se sigue de la *pipe*. Al tratarse de un fichero cuya semántica corresponde a la de un fichero *CSV*, hemos de indicar el delimitador mediante la *flag* `-d:`, y por último seleccionamos con la siguiente *flag* `-f1` el campo que nos interesa del fichero `/etc/passwd`, siendo este el primero pues corresponde al nombre del usuario.

Solución Alternativa:

```
sudo passwd -S -a | awk '$2 ~ /L/ {print $1}'
```

Explicación:

- `sudo passwd -S -a:` Muestra la información sobre todos los usuarios en el sistema, incluyendo su estado de contraseña, gracias a las *flags* `-a` y `-S`, respectivamente.
- `awk '$2 ~ /L/ {print $1}':` Si la segunda columna es "L" significa que la cuenta está bloqueada y no puede iniciar sesión. En la primera columna se muestra el nombre de usuario.

Validación: Al igual que el *Ejercicio 2*, nos creamos un fichero de prueba donde almacenamos 3 usuarios de los cuales dos de ellos no pueden iniciar sesión. Mostramos el contenido del fichero de prueba y por último aplicamos el mismo comando con el que resolvimos el ejercicio, apareciendo por pantalla únicamente aquellos que no pueden iniciar sesión.

2.4. Ejercicio 5

Enunciado: **Muestre los distintos tipos de inicio de sesión de los usuarios que están dados de alta en el sistema, así como el número de usuarios que tienen cada tipo. Ordene dicha información por el número de usuarios.**

Solución:

```
cut -d: -f7 /etc/passwd | sort | uniq -c | sort -nr
```

Explicación: Dado que ya se ha explicado el funcionamiento del comando `cut`, se procederá con la explicación de los nuevos comandos: `sort` y `uniq`. El primero ordena los resultados del fichero especificado según que *flags* se indiquen y el segundo omite aquellos resultados repetidos por pantalla. Siguiendo la naturaleza del ejercicio pedido, tendríamos:

- `cut -d: -f7 /etc/passwd | sort:` Seleccionamos el séptimo campo del fichero `/etc/passwd`, cuya correspondencia es el tipo de inicio de sesión del que dispone el usuario (recordemos que se trata de un fichero *CSV*, de ahí el delimitador `-d:`), y ordenamos la búsqueda.
- `uniq -c:` La *flag* `-c` añade por pantalla el número de ocurrencias repetidas del registro.
- `sort -nr:` Finalmente, ordenamos los tipos de inicio de sesión por número de usuarios (es decir, número de ocurrencias) y en orden descendente, gracias a las *flags* `-nr`, respectivamente.

Solución Alternativa:

```
cat /etc/passwd | awk -F: '{print $NF}' | sort | uniq -c | sort -nr
```

Explicación: Esta alternativa plantea otro medio para obtener la información gracias al uso del comando `cat`, seguido del comando `awk`.

- `cat /etc/passwd | awk -F:` '{`print $NF`}': Copia el contenido del fichero `/etc/passwd` y lo muestra por pantalla mediante `awk`, un lenguaje de programación especializado en la manipulación textual de datos. Con la *flag* `-F` definimos el separador `:` e invocamos seguidamente la función `print` bajo la variable especial `NF`, cuyo propósito es mostrar el último de los campos de la instancia actual.
- El resto de comandos en las siguientes *pipes* son idénticos a la propuesta anterior.

Validación: Se ha creado un archivo `passwd_prueba.txt` donde se insertan 3 usuarios ficticios, dos de ellos con inicio de sesión `/bin/bash` y uno de ellos con `/usr/sbin/nologin`. Se hace uso de `diff -q` para ver que no haya diferencias entre usar el comando del ejercicio con ese archivo y un `echo` con el resultado esperado. Para más claridad se muestran por pantalla el resultado junto al esperado.

3. Procesos

3.1. Ejercicio 6

ENUNCIADO: **Encuentre los programas con el setgid activado y guarde el resultado en *programassetgid.txt.***

SOLUCIÓN:

```
sudo find / -perm /g=s -executable > programassetgid.txt
```

EXPLICACIÓN: Dado que ya se ha explicado el funcionamiento del comando `find`, se procederá con la explicación de las *flags* utilizadas:

- `-perm /g=s -executable`: Buscamos archivos con los permisos especificados `/g=s`, argumento que establece que los archivos buscados han de tener permisos establecidos para el grupo, y que sean ejecutables (`-executable`).
- `> programassetgid.txt`: Redireccionamos la salida de la búsqueda realizada al archivo de texto especificado, guardándolo en el directorio actual.

VALIDACIÓN: Leemos el fichero `programassetgid.txt` con `cat`, y ejecutamos `ls -l` que mostrará el formato largo de listado de todos los programas que tenga el `setgid` activado en el sistema.

3.2. Ejercicio 7

ENUNCIADO: **Identifique los 3 procesos que se estén ejecutando con permisos que no sean de root y que requieran menos memoria.**

SOLUCIÓN:

```
ps haux --sort=-%mem | grep -v root | tail -n 3
```

EXPLICACIÓN: El comando `ps` se utiliza para informar sobre el estado de los procesos que se están ejecutando en el sistema, y con las *flags* `haux --sort=-%mem` indicamos, respectivamente:

- `h`: Que no muestre los encabezados, omitiendo las etiquetas.
- `a`: Selecciona todos los procesos. El uso de la *flag* `x` indica que se listen todos los procesos identificados.
- `u`: Se muestra en formato usuario, es decir, la información se dispone detalladamente al usuario propietario sobre los procesos.
- `--sort=-%mem`: La salida se ordenará de forma descendente en función del porcentaje de uso de memoria.

Seguidamente, tendríamos comando previamente vistos, los cuales, en resumen:

- `grep -v root`: Filtramos la salida del comando previo, usando la *flag* `-v` que realiza la inversa, lo que quiere decir que mostrará todas aquellas líneas que no contengan el patrón especificado.
- `tail -n 3`: Mostramos las últimas líneas de la salida del comando anterior.

VALIDACIÓN: Para validar este ejercicio debemos comprobar 2 cosas:

- Que los procesos encontrados no tengan permisos de `root`.
- Que son los tres procesos que requieren menos memoria.

Para la primera condición, nos basta con guardar la salida del comando `ps aux --sort=+%mem | grep -v root | head -n 4 | grep` "`root`" en una variable, el cual busca la palabra "root" dentro del comando de la solución, añadiendo la fila que especifica los nombres de las columnas, si esta variable es nula, entonces no se encontraron permisos de `root` y tuvimos éxito.

Para la segunda condición existe un problema lógico: hay más de tres procesos cuyo uso de CPU es 0.0, por lo que el comando devolverá los tres primeros que encuentre, y no los que menos CPU usen. Tampoco tenemos forma de comprobar cuáles son así que, tras hablar con el profesor, me aconsejó mostrar el uso de CPU de estos tres procesos para que el usuario pueda comprobar que se trata de valores nulos o cercanos a cero. Esto lo hacemos añadiendo al comando solución `awk` '`{print $3}`', lo cual imprimirá sólo la tercera columna –uso de CPU–.

3.3. Ejercicio 8

ENUNCIADO: **Monitorice la carga del sistema usando el valor de la carga promedio en los últimos 5 minutos. Utilice watch para mostrar dicho valor cada cinco segundos.**

SOLUCIÓN: Proponemos dos soluciones: usando el comando `cut` y usando el comando `awk`.

```
watch -n 5 "cut -d' ' -f 2 /proc/loadavg"
```

EXPLICACIÓN: El comando `watch` ejecuta un programa periodicamente, en función de las *flags* indicadas. En este caso:

- `-n 5`: Es la tasa de actualización; muestra la información cada 5 segundos.
- "`cut -d' ' -f 2 /proc/loadavg`": Comando a ejecutar y cuyo resultado se mostrará por pantalla. En este caso, el comando utiliza `cut` para extraer la segunda columna del archivo `/proc/loadavg`, que representa la carga promedio en los últimos 5 minutos. Indica como delimitador un espacio en blanco con `-d` y selecciona el segundo campo con `-f 2`.

Con el comando awk sería de la siguiente forma:

```
watch -n 5 "cat /proc/loadavg | awk '{print \$2}'"
```

EXPLICACIÓN: La diferencia con respecto a la solución propuesta inicialmente es el medio para mostrar por pantalla. Con el comando `cat` muestra el contenido del fichero `/proc/loadavg` y mediante el uso de `awk` mostramos el contenido haciendo uso de `print`, el cual especifica que se debe imprimir la segunda columna del archivo.

VALIDACIÓN: Nos creamos un fichero de prueba en el que agregamos una línea que añade una carga ficticia del fichero `/proc/loadavg`. Sabemos, mediante `watch -n 5` que la carga promedio es de 0.04, y tras un uso de `cut` al segundo campo del fichero, vemos que coinciden las cargas del sistema.

4. Gestión de usuarios

4.1. Ejercicio 9

Enunciado: **Cree un usuario llamado con sus iniciales. El usuario tendrá home, bash y podrá iniciar sesión.**

Solución:

```
sudo useradd -m -s "/bin/bash" AAT && sudo passwd AAT
```

Explicación: Debido a que el comando `useradd` necesita de permisos de superusuario, usamos `sudo` para que se ejecute correctamente. La naturaleza del comando crea un usuario especificado, en este caso `'ATT'`, mientras que `passwd` le establece una contraseña, que es `'ATT'`:

- `-m -s` "`/bin/bash`": La primera opción crea un directorio de inicio para el usuario, mientras que la segunda opción establece la *shell* que tendrá el usuario.
- El operador `&&` se utiliza para ejecutar el segundo comando solo si el primero se ejecuta correctamente –operador lógico AND–.

Validación: Realizamos las siguientes comprobaciones:

- Que el usuario exista en el fichero `/etc/passwd` –en nuestro script correspondería con el usuario `AAT`–.
- Que el usuario previo tenga un directorio casa.

A continuación, y con las credenciales en pantalla, comprobamos que podemos iniciar sesión con `ATT`.

4.2. Ejercicio 10

Enunciado: **Cree un usuario llamado webuser sin bash, ni home y que no pueda iniciar sesión.**

Solución:

```
sudo useradd -M -s "/bin/nologin" webuser
```

Explicación: A diferencia del anterior, excluyendo el hecho de que estamos creando un usuario, hacemos uso de dos *flags* para alterar el significado del ejercicio anterior, donde:

- `-M`: Evita la creación del directorio de inicio del usuario.
- `-s` "`/bin/nologin`": Especifica el *shell* por defecto que se utilizará para el usuario. En este caso, el fichero especificado se utiliza para evitar que el usuario inicie sesión en el sistema.

Validación: Igual al ejercicio previo, comprobamos si `webuser` existe en el fichero `/etc/passwd`, intentamos entrar al home dicho usuario –cosa que no podremos pues no existe– y por último iniciamos sesión con dicho usuario, dando como resultado un inicio de sesión imposible, pues dicho usuario no puede iniciar sesión.

4.3. Ejercicio 11

Enunciado: **Cree un usuario llamado andres que se pueda identificar como webuser (usando sudo) pero no como root.**

Solución:

```
sudo useradd -m -d /home/andres -s /bin/bash andres
sudo passwd andres
sudo visudo
#Añadir en '#User Privilege Specification':
andres ALL=/bin/su webuser
```

Explicación: Primero, creamos el usuario `andres` al que disponemos de un directorio home –`/home/andres`–, *shell* predeterminada `/bin/bash` y también le asignamos una contraseña con `passwd`, que en nuestro caso le hemos asignado `andres`.

A continuación, deberemos de otorgarle al usuario `andres` permisos de `sudo` identificándose como `webuser`. Para ello, usaremos el comando `sudo visudo` accedemos al fichero `/etc/sudoers`, un archivo de configuración que define quiénes pueden ejecutar comandos con privilegios de superusuario –root– mediante el comando `'sudo'`. La línea añadida `andres ALL=/bin/su webuser` otorga permiso al usuario `andres` para ejecutar el comando `su webuser`, es decir, para loggearse como webuser, en cambio, no podrá loggearse como root ni como ningún otro usuario.

Debido a que `webuser` no tiene directorio casa ni *shell* asociada, `andres` no podrá ejecutar ningún comando como este aun teniendo permiso, es por esto que debemos otorgarle un shell y un home a webuser previamente.

Validación: Primero se comprueba que los dos usuarios (andres y webuser) existan, seguídamente, hacemos login en andres para comprobar el resto de puntos.

Una vez dentro de andres, deberemos ejecutar los comandos impresos que harán lo siguiente:

1º. Intento de login en webuser, lo cual resultará exitoso y dentro del cual deberemos ejecutar exit para continuar la ejecución de andres.sh.

2º. Intento de login en root, lo cual dará error, comprobando que no se puede logear como tal

5. Sistema de ficheros

5.1. Ejercicio 12

ENUNCIADO: **Cree un fichero lleno de ceros, con un tamaño de 100MiB y llámelo *misistemadeficheros.***

SOLUCIÓN:

```
dd if=/dev/zero of=misistemadeficheros bs=1MiB count=100
```

EXPLICACIÓN: El comando `dd` se utiliza para copiar y convertir archivos. A continuación, las *flags* especificadas:

- `if=/dev/zero:` Especifica la ubicación del archivo de entrada. En este caso, el archivo de entrada es "`/dev/zero`", un archivo especial que contiene una secuencia infinita de ceros.
- `of=misistemadeficheros:` Especifica la ubicación y el nombre del archivo de salida que se va a crear.
- `bs=1MiB:` Establece el tamaño del bloque de entrada y salida en 1 MiB. Esto significa que `dd` copiará los datos en bloques de 1 MiB de tamaño.
- `count=100:` Especifica el número de bloques de entrada que se copiarán. En este caso, se copiarán 100 bloques de 1 MiB cada uno.

VALIDACIÓN: El comando `ls -l` mostrará información sobre los ficheros, así como su tamaño en *bytes* en la quinta columna. Se comprueba que dicho tamaño sea 100MiB (104857600 bytes). Para comprobar que efectivamente nuestro fichero está lleno de ceros, lo comparamos hasta su número de bytes con el fichero `/dev/zero`.

5.2. Ejercicio 13

ENUNCIADO: **Cree un sistema de ficheros, usando el fichero del ejercicio anterior, con el formato ext3.**

SOLUCIÓN:

```
mkfs.ext3 misistemadeficheros
```

EXPLICACIÓN: El comando `mkfs` permite crear un sistema de ficheros con un formato determinado, con la extensión `.ext3` definimos dicho formato, que podría ser `ext4` también. Para el formato ext2 usamos el comando `mke2fs`

VALIDACIÓN: Para validar el formato del sistema de fcheros utilizamos el comando `file`, el cual nos mostrará información relevante sobre el mismo. En esta salida buscaremos con `grep` alguna coincidencia

- `1.` Comando `file misistemadeficheros` mostrará información relevante sobre el mismo.
- `2.` Comando `grep ext3` buscará coincidencias de `ext3` en la salida del comando anterior.
- `3.` Guardando el resultado de `grep` en una variable comprobaremos si está vacía o no, en caso afirmativo el formato es otro.

5.3. Ejercicio 14

Enunciado: **Monte el sistema de ficheros que acaba de crear en el punto de montaje /mimontaje.**

Solución:

```
sudo mkdir /mimontaje
sudo mount misistemadeficheros /mimontaje
```

Explicación: Un punto de montaje es sólo un directorio del que cuelgan otros directorios y/o archivos, por lo que sólo tenemos que crear el directorio `mimontaje` con `mkdir`. Seguídamente, usamos el comando `mount` para montar `misistemadefichero` bajo `mimontaje`.

Validación: Usando el comando `mount` sin argumentos obtenemos una lista de los montajes del sistema, dentro de ella buscamos la frase `misistemadeficheros on /mimontaje` y comprobamos que cuelga de `mimontaje`

6. Bash scripting

6.1. Ejercicio 15

ENUNCIADO: **Desarrolle un programa que permita cambiar el nombre de los ficheros de un directorio dado como argumento de entrada. El nuevo nombre será un número secuencial, manteniendo la extensión.**

SOLUCIÓN:

```
#!/bin/bash

directorio=$1
cont=0


for archivo in $(ls $directorio); do
        cont=$((cont+1))
        extension=$(basename "$archivo" | cut -d. -f2)
        mv $directorio/$archivo $directorio/$cont.$extension

done
```

EXPLICACIÓN: El comando `basename` sin pasarle ningún sufijo devolverá el nombre completo del archivo, del que nos quedamos con la extensión –gracias al comando `cut`– y la guardamos en la variable "extensión". Renombramos entonces los archivos del directorio recibido como parámetro asignándoles de nombre el valor de "contador" –por cada archivo del directorio se incrementa– y de extensión la suya original almacenada en la variable "extensión".

VALIDACIÓN: Hemos creado un directorio de prueba donde se han creado algunos archivos de diferentes extensiones. La idea es ejecutar un comando que muestre por salida los tipos de extensión y cuente cuántos ficheros de cada extensión hay antes y después de su ejecución. El *script* del *ejercicio 15* pasará la validación si las dos salidas mencionadas son iguales, y si los nombres de todos los ficheros tras la ejecución del script son numéricos –se ha empleado una expresión regular–.

6.2. Ejercicio 16

ENUNCIADO: **Se le pide desarrollar un programa que realice una copia de los directorios que cuelguen de /importante/ en /media/backup. Cada directorio que cuelga de /importante/ deberá guardarse en un fichero .tar.xz .**

SOLUCIÓN:

```
#!/bin/bash

mkdir /vagrant/media
mkdir /vagrant/media/backup

mkdir ~/importante
mkdir ~/importante/directorio1
```

```
mkdir ~/importante/directorio2
mkdir ~/importante/directorio3

touch ~/importante/directorio1/archivo1.txt
touch ~/importante/directorio2/archivo1.txt
touch ~/importante/directorio3/archivo1.txt

fecha=$(date +%Y%m%d)
for directorio in $(sudo ls ~/importante); do
        sudo tar -cJf /vagrant/media/backup/"$directorio"_"$fecha".tar.xz ~/importante/"$directorio"
done
```

EXPLICACIÓN: Creamos el directorio ∼/importante con sus subdirectorios, y metemos archivos .txt para que no estén vacíos. Guardamos en una variable la fecha del sistema en formato año, mes y día gracias al comando date. El comando tar nos permite comprimir directorios. Las opción -c permite que como resultado de la ejecución se cree un nuevo archivo. La opción -J permite que la extensión sea .tar.xz. Con la opción -f se especifican los nombres de los ficheros destino o resultado.

VALIDACIÓN: Para la comprobación eliminamos las carpetas que se crean en el *script* del ejercicio previamente a la ejecución del mismo. Tras ejecutarse, se comprueba en la carpeta resultante /vagrant/media/backup que contiene todos y cada uno de los archivos de ∼/importante con el formato indicado en el enunciado gracias a los comandos ls y grep con la opcion -q, que devuelve cero si grep da resultado.

6.3. Ejercicio 17

ENUNCIADO: **Modifique el programa anterior, de tal modo de que para cada directorio se guarden sólo los 5 últimos ficheros. Es decir, al copiar el sexto fichero asociado a un mismo directorio, se borrará la copia más antigua.**

SOLUCIÓN:

```
#!/bin/bash

mkdir /vagrant/media
mkdir /vagrant/media/backup

mkdir ~/importante
mkdir ~/importante/directorio1
mkdir ~/importante/directorio2
mkdir ~/importante/directorio3

touch ~/importante/directorio1/archivo1.txt
touch ~/importante/directorio1/archivo2.txt
touch ~/importante/directorio1/archivo3.txt
touch ~/importante/directorio1/archivo4.txt
touch ~/importante/directorio1/archivo5.txt
touch ~/importante/directorio1/archivo6.txt
```

```
touch ~/importante/directorio2/archivo1.txt
touch ~/importante/directorio2/archivo2.txt
touch ~/importante/directorio2/archivo3.txt
touch ~/importante/directorio2/archivo4.txt
touch ~/importante/directorio2/archivo5.txt
touch ~/importante/directorio2/archivo6.txt

touch ~/importante/directorio3/archivo1.txt
touch ~/importante/directorio3/archivo2.txt
touch ~/importante/directorio3/archivo3.txt
touch ~/importante/directorio3/archivo4.txt
touch ~/importante/directorio3/archivo5.txt
touch ~/importante/directorio3/archivo6.txt

fecha=$(date +%Y%m%d)
for directorio in $(sudo ls ~/importante); do
    archivos=$(ls -t ~/importante/"$directorio" | tail -n +6)

    if [[ -n "$archivos" ]]; then
        sudo rm -f ~/importante/"$directorio"/$archivos
    fi

    sudo tar -cJf /vagrant/media/backup/"$directorio"_"$fecha".tar.xz ~/importante/"$directorio"
done
```

EXPLICACIÓN: Dado que se trata de una modificación del ejercicio anterior, nos remitiremos a explicar las diferencias apreciadas en el código bash, donde:

- Creamos el suficiente número de archivos para realizar la comprobación de, al menos, más de cinco archivos por directorio.
- `archivos=$(ls -t ~/importante/"$directorio"| tail -n +6)`: Obtiene la lista de archivos del directorio "$directorio" en la carpeta "/importante" y la ordena por fecha de modificación, para después eliminar los más antiguos si hay más de cinco archivos en el directorio –`tail -n +6`–.
- En el bucle `if` se comprueba si hay más de cinco archivos en el directorio ~`/importante/`"`$directorio`". En tal caso, la línea a ejecutar eliminará los más antiguos.

VALIDACIÓN: Partiendo de nuestro *script* de comprobación del ejercicio previo, añadimos una anidación de dos bucles `if` la cual comprueba que se hayan realizado correctamente las acciones de eliminación de archivos antiguos y creación de archivos de `backup` para cada uno de los directorios dentro de ~`/importante`.

6.4. Ejercicio 18

Enunciado: **Utilice crontab para ejecutar el programa de backup cada día a las 9 y media de la mañana.**

Solución:

```
#Nos creamos un script de bash, con nombre ejercicio18.sh, que incluya la ruta
#absoluta de nuestro backup:
/vagrant/media/backup
#Editamos el fichero crontab con:
crontab -e
#Seleccionamos editor de texto y agregamos al final del fichero la siguiente línea
#y guardamos al salir.
30 9 * * * /bin/bash /vagrant/ejercicio18.sh
```

Explicación: Explicaremos concretamente la línea agregada al fichero `crontab`:

- `30 9 * * *`: El primer campo especifica los minutos, el segundo la hora, el tercer, cuarto y quinto campo especifican el día del mes, mes y día de la semana, respectivamente. Al usar `*`, estamos indicando que se hagan en todos los días del mes, cada mes y cada día de la semana.
- `/bin/bash /vagrant/ejercicio18.sh`: El último campo especifica el comando que se debe ejecutar en el momento programado.

Validación: La idea principal del *script* es comprobar que el contenido de `ejercicio18.sh` se corresponde con la última línea agregada al fichero `crontab`, que es la instrucción la pedida por el ejercicio. En caso de que coincidan, significará que la tarea para realizar el *backup* con las condiciones descritas está incluída.